import React, { useState } from 'react';

import { Users, Building2, Upload, Download, ArrowRight, FileText, Mail, Plus, X } from 'lucide-react';

import { useData } from '../../context/DataContext';

import { BarChart, Bar, XAxis, YAxis, Tooltip, Legend, ResponsiveContainer } from 'recharts';

interface CSVStudent {

  name: string;

  rollNumber: string;

  year: number;

  section: string;

}

interface SelectedClassroom {

  id: string;

  name: string;

  type: 'room' | 'seminar\_hall';

  capacity: number;

  config: {

    numberOfStudents: number;

    numberOfBenches?: number;

    studentsPerBench?: number;

    numberOfRows?: number;

    numberOfColumns?: number;

  };

}

interface ExamSelection {

  year1Exam: string;

  year2Exam: string;

  year3Exam: string;

  year4Exam: string;

}

const ALGO\_OPTIONS = [

  { value: 'greedy', label: 'Greedy Graph Coloring (O(V+E))' },

  { value: 'randomized', label: 'Randomized Assignment (O(n))' },

  { value: 'simulated', label: 'Simulated Annealing (Heuristic)' }

];

const SeatingArrangements: React.FC = () => {

  const { exams, classrooms, addSeatingArrangement } = useData();

  const [examSelections, setExamSelections] = useState<ExamSelection>({

    year1Exam: '',

    year2Exam: '',

    year3Exam: '',

    year4Exam: ''

  });

  const [selectedClassrooms, setSelectedClassrooms] = useState<SelectedClassroom[]>([]);

  const [csvStudents, setCsvStudents] = useState<CSVStudent[]>([]);

  const [generatedSeating, setGeneratedSeating] = useState<any[]>([]);

  const [step, setStep] = useState<'upload' | 'config' | 'generate'>('upload');

  const [selectedAlgo, setSelectedAlgo] = useState('greedy');

  const [algoInfo, setAlgoInfo] = useState<{ name: string; description: string; complexity: string; runtime: number } | null>(null);

  const [loading, setLoading] = useState(false);

  const handleCSVUpload = (event: React.ChangeEvent<HTMLInputElement>) => {

    const file = event.target.files?.[0];

    if (!file) return;

    const reader = new FileReader();

    reader.onload = (e) => {

      const text = e.target?.result as string;

      const lines = text.split('\n').filter(line => line.trim());

      const headers = lines[0].split(',').map(h => h.trim().toLowerCase());

      const students: CSVStudent[] = [];

      for (let i = 1; i < lines.length; i++) {

        const values = lines[i].split(',').map(v => v.trim());

        if (values.length >= 4) {

          students.push({

            name: values[headers.indexOf('name')] || values[0],

            rollNumber: values[headers.indexOf('rollnumber')] || values[1],

            year: parseInt(values[headers.indexOf('year')] || values[2]) || 1,

            section: values[headers.indexOf('section')] || values[3]

          });

        }

      }

      setCsvStudents(students);

      setStep('config');

    };

    reader.readAsText(file);

  };

  const downloadSampleCSV = () => {

    const sampleData = [

      'Name,RollNumber,Year,Section',

      'John Smith,CS2021001,3,A',

      'Alice Johnson,CS2022001,2,B',

      'Bob Wilson,CS2023001,1,A',

      'Carol Davis,CS2021002,3,B'

    ].join('\n');

    const blob = new Blob([sampleData], { type: 'text/csv' });

    const url = window.URL.createObjectURL(blob);

    const a = document.createElement('a');

    a.href = url;

    a.download = 'sample\_students.csv';

    a.click();

    window.URL.revokeObjectURL(url);

  };

  const addClassroom = () => {

    const availableClassrooms = classrooms.filter(c =>

      c.available && !selectedClassrooms.some(sc => sc.id === c.id)

    );

    if (availableClassrooms.length > 0) {

      const classroom = availableClassrooms[0];

      const newSelectedClassroom: SelectedClassroom = {

        id: classroom.id,

        name: classroom.name,

        type: classroom.type,

        capacity: classroom.capacity,

        config: {

          numberOfStudents: 0,

          ...(classroom.type === 'room' ? {

            numberOfBenches: classroom.benches || 15,

            studentsPerBench: classroom.benchCapacity || 2

          } : {

            numberOfRows: classroom.rows || 10,

            numberOfColumns: classroom.columns || 10

          })

        }

      };

      setSelectedClassrooms([...selectedClassrooms, newSelectedClassroom]);

    }

  };

  const removeClassroom = (index: number) => {

    setSelectedClassrooms(selectedClassrooms.filter((\_, i) => i !== index));

  };

  const updateClassroomConfig = (index: number, config: any) => {

    const updated = [...selectedClassrooms];

    updated[index].config = { ...updated[index].config, ...config };

    setSelectedClassrooms(updated);

  };

  const changeClassroom = (index: number, newClassroomId: string) => {

    const classroom = classrooms.find(c => c.id === newClassroomId);

    if (!classroom) return;

    const updated = [...selectedClassrooms];

    updated[index] = {

      id: classroom.id,

      name: classroom.name,

      type: classroom.type,

      capacity: classroom.capacity,

      config: {

        numberOfStudents: 0,

        ...(classroom.type === 'room' ? {

          numberOfBenches: classroom.benches || 15,

          studentsPerBench: classroom.benchCapacity || 2

        } : {

          numberOfRows: classroom.rows || 10,

          numberOfColumns: classroom.columns || 10

        })

      }

    };

    setSelectedClassrooms(updated);

  };

  // Utility: Check if a student can be placed at a given seat (no adjacent same year)

  function canPlaceStudent(seatingGrid: any[][], row: number, col: number, student: CSVStudent) {

    const directions = [

      [0, -1], // left

      [0, 1],  // right

      [-1, 0], // up/front

      [1, 0]   // down/back

    ];

    for (const [dr, dc] of directions) {

      const r = row + dr, c = col + dc;

      if (r >= 0 && r < seatingGrid.length && c >= 0 && c < seatingGrid[0].length) {

        const adj = seatingGrid[r][c];

        if (adj && adj.year === student.year) return false;

      }

    }

    return true;

  }

  const generateSeatingArrangement = async () => {

    if (selectedClassrooms.length === 0 || csvStudents.length === 0) return;

    setLoading(true);

    setAlgoInfo(null);

    const t0 = performance.now();

    await new Promise(res => setTimeout(res, 100)); // UI delay

    // Determine exam logic based on selections

    const selectedExams = Object.values(examSelections).filter(exam => exam !== '');

    const isMixedYear = selectedExams.length > 1;

    const isSingleYear = selectedExams.length === 1;

    // Sort classrooms: rooms first, then seminar halls

    const sortedClassrooms = [...selectedClassrooms].sort((a, b) => {

      if (a.type === 'room' && b.type === 'seminar\_hall') return -1;

      if (a.type === 'seminar\_hall' && b.type === 'room') return 1;

      return 0;

    });

    const allSeating: any[] = [];

    let remainingStudents = [...csvStudents];

    // Group students by year for better distribution

    const yearGroups: Record<number, CSVStudent[]> = {};

    remainingStudents.forEach(s => {

      if (!yearGroups[s.year]) yearGroups[s.year] = [];

      yearGroups[s.year].push(s);

    });

    const years = Object.keys(yearGroups).map(Number).sort();

    let algoName = '';

    let algoDesc = '';

    let algoComplexity = '';

    for (const classroom of sortedClassrooms) {

      if (remainingStudents.length === 0) break;

      const studentsForThisRoom = remainingStudents.slice(0, classroom.config.numberOfStudents);

      remainingStudents = remainingStudents.slice(classroom.config.numberOfStudents);

      const seating: any[] = [];

      if (classroom.type === 'room') {

        // Room seating arrangement with algorithm selection

        const { numberOfBenches, studentsPerBench } = classroom.config;

        const benchGrid: (CSVStudent|null)[][] = Array.from({ length: numberOfBenches! }, () => Array(studentsPerBench!).fill(null));

        if (selectedAlgo === 'greedy' || selectedAlgo === 'randomized' || selectedAlgo === 'simulated') {

          algoName = `${selectedAlgo.charAt(0).toUpperCase() + selectedAlgo.slice(1)} Bench Assignment (Strict No-Adjacent-Year)`;

          algoDesc = 'Assigns students to benches so that no two adjacent (left, right, front, back, or same bench) are from the same year.';

          algoComplexity = selectedAlgo === 'greedy' ? 'O(n^2)' : selectedAlgo === 'randomized' ? 'O(n^2)' : 'O(n^2)';

          // Update year groups for this classroom

          const localYearGroups: Record<number, CSVStudent[]> = {};

          studentsForThisRoom.forEach(s => {

            if (!localYearGroups[s.year]) localYearGroups[s.year] = [];

            localYearGroups[s.year].push(s);

          });

          const localYears = Object.keys(localYearGroups).map(Number).sort();

          // CRITICAL FIX: Check if we actually have multiple years in the data

          const actualYearsInData = [...new Set(studentsForThisRoom.map(s => s.year))];

          const hasMultipleYears = actualYearsInData.length > 1;

          if (isSingleYear || !hasMultipleYears) {

            // Same year students - apply bench capacity rules

            const shuffledStudents = [...studentsForThisRoom].sort(() => Math.random() - 0.5);

            let studentIndex = 0;

            for (let bench = 0; bench < numberOfBenches!; bench++) {

              if (studentsPerBench === 1) {

                // Only one seat per bench, alternate benches for spacing

                if (bench % 2 === 0 && studentIndex < shuffledStudents.length) {

                  benchGrid[bench][0] = shuffledStudents[studentIndex++];

                }

              } else if (studentsPerBench === 2) {

                // Only first seat used for single year (single occupancy)

                if (studentIndex < shuffledStudents.length) {

                  benchGrid[bench][0] = shuffledStudents[studentIndex++];

                }

              } else if (studentsPerBench === 3) {

                // Place at ends (seats 0 and 2), leave middle empty

                if (studentIndex < shuffledStudents.length) {

                  benchGrid[bench][0] = shuffledStudents[studentIndex++];

                }

                if (studentIndex < shuffledStudents.length) {

                  benchGrid[bench][2] = shuffledStudents[studentIndex++];

                }

                // seat 1 (middle) left empty

              } else if (studentsPerBench && studentsPerBench >= 4) {

                // Place at first and last seat

                if (studentIndex < shuffledStudents.length) {

                  benchGrid[bench][0] = shuffledStudents[studentIndex++];

                }

                if (studentIndex < shuffledStudents.length) {

                  benchGrid[bench][studentsPerBench - 1] = shuffledStudents[studentIndex++];

                }

                // all other seats left empty

              }

            }

          } else {

            // Multiple years - apply special bench rules for 2 and 3 capacity, else use standard mixing

            if (studentsPerBench === 2) {

              // FIXED: Place students from DIFFERENT YEARS only

              let benchIdx = 0;

              const availableYears = localYears.filter(year => localYearGroups[year].length > 0);

              while (benchIdx < numberOfBenches! && availableYears.length >= 2) {

                // Find two different years with available students

                let year1 = availableYears[0];

                let year2 = availableYears.find(y => y !== year1);

                if (!year2) break; // No different year available

                let student1 = localYearGroups[year1].length > 0 ? localYearGroups[year1].shift() : null;

                let student2 = localYearGroups[year2].length > 0 ? localYearGroups[year2].shift() : null;

                // Only place if we have students from DIFFERENT years

                if (student1 && student2 && student1.year !== student2.year) {

                  benchGrid[benchIdx][0] = student1;

                  benchGrid[benchIdx][1] = student2;

                  benchIdx++;

                } else {

                  break; // Can't find different year students

                }

                // Update available years list

                availableYears.splice(0, availableYears.length);

                localYears.forEach(year => {

                  if (localYearGroups[year].length > 0) {

                    availableYears.push(year);

                  }

                });

              }

              // If any students left and we have space, fill remaining seats with same year logic

              let flatLeft = ([] as CSVStudent[]).concat(...Object.values(localYearGroups));

              for (; benchIdx < numberOfBenches! && flatLeft.length > 0; benchIdx++) {

                // For remaining students, use single occupancy (same year logic)

                if (flatLeft.length > 0) {

                  benchGrid[benchIdx][0] = flatLeft.shift() || null;

                }

              }

            } else if (studentsPerBench === 3) {

              // Place different year in middle, same years at ends

              let benchIdx = 0;

              while (benchIdx < numberOfBenches!) {

                // Try to get 2 same year and 1 different year

                let y1 = localYears[0], y2 = localYears[1] || localYears[0];

                let s1 = (localYearGroups[y1] && localYearGroups[y1].length > 0) ? localYearGroups[y1].shift() : null;

                let s2 = (localYearGroups[y1] && localYearGroups[y1].length > 0) ? localYearGroups[y1].shift() : null;

                let s3 = (localYearGroups[y2] && localYearGroups[y2].length > 0) ? localYearGroups[y2].shift() : null;

                // Place same year at ends, different year in middle

                benchGrid[benchIdx][0] = s1;

                benchGrid[benchIdx][1] = s3;

                benchGrid[benchIdx][2] = s2;

                benchIdx++;

              }

              // If any students left, fill remaining seats

              let flatLeft = ([] as CSVStudent[]).concat(...Object.values(localYearGroups));

              let seatPtr = 0;

              for (; benchIdx < numberOfBenches! && flatLeft.length > 0; benchIdx++) {

                for (let seat = 0; seat < 3 && flatLeft.length > 0; seat++) {

                  benchGrid[benchIdx][seat] = flatLeft.shift() || null;

                }

              }

            } else {

              // Standard year-mixing algorithm for other capacities

              for (let bench = 0; bench < numberOfBenches!; bench++) {

                for (let seat = 0; seat < studentsPerBench!; seat++) {

                  let placed = false;

                  for (let y = 0; y < localYears.length; y++) {

                    const year = localYears[(bench + seat + y) % localYears.length];

                    if (localYearGroups[year].length === 0) continue;

                    const candidate = localYearGroups[year][0];

                    if (canPlaceStudent(benchGrid, bench, seat, candidate)) {

                      benchGrid[bench][seat] = candidate;

                      localYearGroups[year].shift();

                      placed = true;

                      break;

                    }

                  }

                  if (!placed) benchGrid[bench][seat] = null;

                }

              }

            }

          }

        }

        // Flatten to seating array

        for (let bench = 0; bench < numberOfBenches!; bench++) {

          for (let seat = 0; seat < studentsPerBench!; seat++) {

            seating.push({

              classroomId: classroom.id,

              classroomName: classroom.name,

              benchNumber: bench + 1,

              seatNumber: seat + 1,

              student: benchGrid[bench][seat],

              position: `${classroom.name}-B${bench + 1}S${seat + 1}`

            });

          }

        }

      } else {

        // Seminar hall seating arrangement with algorithm selection

        const { numberOfRows, numberOfColumns } = classroom.config;

        const grid: (CSVStudent|null)[][] = Array.from({ length: numberOfRows! }, () => Array(numberOfColumns!).fill(null));

        if (selectedAlgo === 'greedy' || selectedAlgo === 'randomized' || selectedAlgo === 'simulated') {

          algoName = `${selectedAlgo.charAt(0).toUpperCase() + selectedAlgo.slice(1)} Grid Assignment (Strict No-Adjacent-Year)`;

          algoDesc = 'Assigns students to seats so that no two adjacent (front, back, left, right) are from the same year.';

          algoComplexity = selectedAlgo === 'greedy' ? 'O(n^2)' : selectedAlgo === 'randomized' ? 'O(n^2)' : 'O(n^2)';

          // Update year groups for this classroom

          const localYearGroups: Record<number, CSVStudent[]> = {};

          studentsForThisRoom.forEach(s => {

            if (!localYearGroups[s.year]) localYearGroups[s.year] = [];

            localYearGroups[s.year].push(s);

          });

          const localYears = Object.keys(localYearGroups).map(Number).sort();

          if (isSingleYear || localYears.length === 1) {

            // Same year students - use checkerboard pattern for spacing

            const shuffledStudents = [...studentsForThisRoom].sort(() => Math.random() - 0.5);

            let studentIndex = 0;

            for (let row = 0; row < numberOfRows!; row++) {

              for (let col = 0; col < numberOfColumns!; col++) {

                const shouldPlaceStudent = (row + col) % 2 === 0;

                const student = shouldPlaceStudent && studentIndex < shuffledStudents.length

                  ? shuffledStudents[studentIndex++]

                  : null;

                grid[row][col] = student;

              }

            }

          } else {

            // Multiple years - use original mixing logic

            for (let row = 0; row < numberOfRows!; row++) {

              for (let col = 0; col < numberOfColumns!; col++) {

                let placed = false;

                for (let y = 0; y < localYears.length; y++) {

                  const year = localYears[(row + col + y) % localYears.length];

                  if (localYearGroups[year].length === 0) continue;

                  const candidate = localYearGroups[year][0];

                  if (canPlaceStudent(grid, row, col, candidate)) {

                    grid[row][col] = candidate;

                    localYearGroups[year].shift();

                    placed = true;

                    break;

                  }

                }

                if (!placed) grid[row][col] = null;

              }

            }

          }

        }

        for (let row = 0; row < numberOfRows!; row++) {

          for (let col = 0; col < numberOfColumns!; col++) {

            seating.push({

              classroomId: classroom.id,

              classroomName: classroom.name,

              row: row + 1,

              column: col + 1,

              student: grid[row][col],

              position: `${classroom.name}-R${row + 1}C${col + 1}`

            });

          }

        }

      }

      allSeating.push(...seating);

    }

    const t1 = performance.now();

    setGeneratedSeating(allSeating);

    setAlgoInfo({ name: algoName, description: algoDesc, complexity: algoComplexity, runtime: Math.round(t1 - t0) });

    setLoading(false);

    setStep('generate');

  };

  const saveSeatingArrangement = () => {

    const selectedExams = Object.values(examSelections).filter(exam => exam !== '');

    if (selectedExams.length > 0 && selectedClassrooms.length > 0 && generatedSeating.length > 0) {

      // Save arrangement for each classroom

      selectedClassrooms.forEach(classroom => {

        const classroomSeating = generatedSeating.filter(seat => seat.classroomId === classroom.id);

        if (classroomSeating.length > 0) {

          // Use the first selected exam for saving (can be enhanced to handle multiple exams)

          addSeatingArrangement({

            examId: selectedExams[0],

            classroomId: classroom.id,

            seats: classroomSeating

          });

        }

      });

      alert('Seating arrangements saved successfully for all classrooms!');

      // Reset form

      setStep('upload');

      setCsvStudents([]);

      setGeneratedSeating([]);

      setExamSelections({ year1Exam: '', year2Exam: '', year3Exam: '', year4Exam: '' });

      setSelectedClassrooms([]);

    }

  };

  const downloadSeatingArrangement = (format: 'csv' | 'pdf') => {

    if (generatedSeating.length === 0) {

      alert('No seating arrangement to download');

      return;

    }

    const selectedExams = Object.values(examSelections).filter(exam => exam !== '');

    // FIXED: Get ALL selected exam details, not just the first one

    const selectedExamDetails = selectedExams.map(examId =>

      exams.find(e => e.id === examId)

    ).filter(exam => exam !== undefined);

    // Create comprehensive title for multiple exams

    const examTitle = selectedExamDetails.length > 1

      ? `Multi-Exam-Arrangement-${selectedExamDetails.map(e => e?.subject).join('-')}`

      : selectedExamDetails[0]?.subject || 'Single-Exam';

    // Create date string from all exam dates

    const examDates = selectedExamDetails.length > 1

      ? selectedExamDetails.map(e => new Date(e?.date || '').toLocaleDateString()).join('-')

      : selectedExamDetails[0] ? new Date(selectedExamDetails[0].date).toLocaleDateString() : 'date';

    if (format === 'csv') {

      const csvContent = [

        'Position,Student Name,Roll Number,Year,Section,Classroom,Bench Number,Seat Number,Row,Column',

        ...generatedSeating

          .filter(seat => seat.student)

          .map(seat => [

            seat.position,

            seat.student?.name || '',

            seat.student?.rollNumber || '',

            seat.student?.year || '',

            seat.student?.section || '',

            seat.classroomName || '',

            seat.benchNumber || '',

            seat.seatNumber || '',

            seat.row || '',

            seat.column || ''

          ].join(','))

      ].join('\n');

      const blob = new Blob([csvContent], { type: 'text/csv' });

      const url = window.URL.createObjectURL(blob);

      const a = document.createElement('a');

      a.href = url;

      a.download = `seating\_arrangement\_${examTitle}\_${examDates.replace(/\//g, '-')}.csv`;

      a.click();

      window.URL.revokeObjectURL(url);

    } else if (format === 'pdf') {

      // FIXED: Show all selected exams in PDF title and details

      const examListHtml = selectedExamDetails.length > 1

        ? selectedExamDetails.map(exam => `

            <div style="margin-bottom: 10px; padding: 8px; background-color: #f8f9fa; border-left: 3px solid #007bff;">

              <strong>Year ${exam?.year}:</strong> ${exam?.subject} - Section ${exam?.section}<br>

              <small>Date: ${exam ? new Date(exam.date).toLocaleDateString() : 'N/A'} | Time: ${exam?.time || 'N/A'}</small>

            </div>

          `).join('')

        : `

            <div style="margin-bottom: 10px;">

              <strong>Subject:</strong> ${selectedExamDetails[0]?.subject || 'N/A'}<br>

              <strong>Year:</strong> ${selectedExamDetails[0]?.year || 'N/A'} |

              <strong>Section:</strong> ${selectedExamDetails[0]?.section || 'N/A'}

            </div>

          `;

      const pdfContent = `

        <!DOCTYPE html>

        <html>

        <head>

          <title>Seating Arrangement - ${selectedExamDetails.length > 1 ? 'Multi-Exam Arrangement' : selectedExamDetails[0]?.subject || 'Exam'}</title>

          <style>

            body { font-family: Arial, sans-serif; margin: 20px; }

            .header { text-align: center; margin-bottom: 30px; }

            .exam-info { margin-bottom: 20px; }

            .exam-list { margin-bottom: 20px; }

            table { width: 100%; border-collapse: collapse; margin-top: 20px; }

            th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }

            th { background-color: #f2f2f2; }

            .multi-exam-badge {

              background-color: #e3f2fd;

              color: #1976d2;

              padding: 4px 8px;

              border-radius: 4px;

              font-size: 12px;

              font-weight: bold;

            }

          </style>

        </head>

        <body>

          <div class="header">

            <h1>Seating Arrangement</h1>

            <h2>

              ${selectedExamDetails.length > 1

                ? `<span class="multi-exam-badge">MULTI-EXAM</span> Combined Arrangement`

                : selectedExamDetails[0]?.subject || 'Single Exam'}

            </h2>

          </div>

          <div class="exam-info">

            <h3>Exam Details:</h3>

            <div class="exam-list">

              ${examListHtml}

            </div>

            <p><strong>Classrooms:</strong> ${selectedClassrooms.map(c => c.name).join(', ')}</p>

            <p><strong>Total Students:</strong> ${generatedSeating.filter(s => s.student).length}</p>

            <p><strong>Algorithm Used:</strong> ${algoInfo?.name || 'N/A'}</p>

            <p><strong>Arrangement Type:</strong> ${selectedExamDetails.length > 1 ? 'Mixed-Year Logic' : 'Same-Year Logic'}</p>

          </div>

          <table>

            <thead>

              <tr>

                <th>Position</th>

                <th>Student Name</th>

                <th>Roll Number</th>

                <th>Year</th>

                <th>Section</th>

                <th>Classroom</th>

              </tr>

            </thead>

            <tbody>

              ${generatedSeating

                .filter(seat => seat.student)

                .map(seat => `

                  <tr>

                    <td>${seat.position}</td>

                    <td>${seat.student?.name || ''}</td>

                    <td>${seat.student?.rollNumber || ''}</td>

                    <td>${seat.student?.year || ''}</td>

                    <td>${seat.student?.section || ''}</td>

                    <td>${seat.classroomName || ''}</td>

                  </tr>

                `).join('')}

            </tbody>

          </table>

        </body>

        </html>

      `;

      const newWindow = window.open('', '\_blank');

      if (newWindow) {

        newWindow.document.write(pdfContent);

        newWindow.document.close();

        newWindow.print();

      }

    }

  };

  const handleSendMail = () => {

    window.open('http://localhost:5174/', '\_blank');

  };

  const totalConfiguredStudents = selectedClassrooms.reduce((sum, classroom) => sum + classroom.config.numberOfStudents, 0);

  const hasValidExamSelection = Object.values(examSelections).some(exam => exam !== '');

  return (

    <div className="p-6">

      <div className="mb-6">

        <h1 className="text-2xl font-bold text-gray-900">Seating Arrangements</h1>

        <p className="text-gray-600 mt-1">Generate and manage exam seating arrangements across multiple classrooms</p>

      </div>

      {/\* Progress Steps \*/}

      <div className="mb-8">

        <div className="flex items-center justify-center space-x-8">

          <div className={`flex items-center ${step === 'upload' ? 'text-blue-600' : step === 'config' || step === 'generate' ? 'text-green-600' : 'text-gray-400'}`}>

            <div className={`w-8 h-8 rounded-full flex items-center justify-center ${step === 'upload' ? 'bg-blue-100' : step === 'config' || step === 'generate' ? 'bg-green-100' : 'bg-gray-100'}`}>

              <Upload className="w-4 h-4" />

            </div>

            <span className="ml-2 font-medium">Upload Students</span>

          </div>

          <ArrowRight className="w-4 h-4 text-gray-400" />

          <div className={`flex items-center ${step === 'config' ? 'text-blue-600' : step === 'generate' ? 'text-green-600' : 'text-gray-400'}`}>

            <div className={`w-8 h-8 rounded-full flex items-center justify-center ${step === 'config' ? 'bg-blue-100' : step === 'generate' ? 'bg-green-100' : 'bg-gray-100'}`}>

              <Building2 className="w-4 h-4" />

            </div>

            <span className="ml-2 font-medium">Configure Classrooms</span>

          </div>

          <ArrowRight className="w-4 h-4 text-gray-400" />

          <div className={`flex items-center ${step === 'generate' ? 'text-blue-600' : 'text-gray-400'}`}>

            <div className={`w-8 h-8 rounded-full flex items-center justify-center ${step === 'generate' ? 'bg-blue-100' : 'bg-gray-100'}`}>

              <Users className="w-4 h-4" />

            </div>

            <span className="ml-2 font-medium">Generate Seating</span>

          </div>

        </div>

      </div>

      {/\* Step 1: Upload CSV \*/}

      {step === 'upload' && (

        <div className="max-w-2xl mx-auto">

          <div className="bg-white rounded-xl shadow-sm border border-gray-200 p-6">

            <h2 className="text-lg font-semibold text-gray-900 mb-4">Upload Student List & Select Exams</h2>

            <div className="space-y-4">

              {/\* Algorithm Selection \*/}

              <div>

                <label className="block text-sm font-medium text-gray-700 mb-2">

                  Select Algorithm <span className="text-xs text-blue-700">(DAA)</span>

                </label>

                <select

                  value={selectedAlgo}

                  onChange={e => setSelectedAlgo(e.target.value)}

                  className="w-full px-3 py-2 border border-blue-400 rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500 mb-2"

                >

                  {ALGO\_OPTIONS.map(opt => (

                    <option key={opt.value} value={opt.value}>{opt.label}</option>

                  ))}

                </select>

              </div>

              {/\* Multi-Exam Selection \*/}

              <div className="bg-blue-50 border border-blue-200 rounded-lg p-4">

                <h3 className="font-medium text-blue-900 mb-3">Exam Selection by Year</h3>

                <p className="text-sm text-blue-800 mb-3">Select exams for different years. Leave empty for no exam for that year.</p>

                <div className="grid grid-cols-1 md:grid-cols-2 gap-4">

                  {[1, 2, 3, 4].map(year => (

                    <div key={year}>

                      <label className="block text-sm font-medium text-blue-700 mb-1">

                        Year {year} Exam

                      </label>

                      <select

                        value={examSelections[`year${year}Exam` as keyof ExamSelection]}

                        onChange={(e) => setExamSelections({

                          ...examSelections,

                          [`year${year}Exam`]: e.target.value

                        })}

                        className="w-full px-3 py-2 border border-blue-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500"

                      >

                        <option value="">No Exam</option>

                        {exams.filter(e => e.status === 'scheduled' && e.year === year).map(exam => (

                          <option key={exam.id} value={exam.id}>

                            {exam.subject} - Section {exam.section}

                          </option>

                        ))}

                      </select>

                    </div>

                  ))}

                </div>

                <div className="mt-3 text-xs text-blue-700">

                  <strong>Logic:</strong> Single exam = same year logic | Multiple exams = mixed year logic

                </div>

              </div>

              <div>

                <label className="block text-sm font-medium text-gray-700 mb-2">

                  Upload CSV File

                </label>

                <div className="border-2 border-dashed border-gray-300 rounded-lg p-6 text-center">

                  <FileText className="w-12 h-12 text-gray-400 mx-auto mb-4" />

                  <p className="text-gray-600 mb-4">Upload a CSV file with student information</p>

                  <input

                    type="file"

                    accept=".csv"

                    onChange={handleCSVUpload}

                    className="hidden"

                    id="csv-upload"

                  />

                  <label

                    htmlFor="csv-upload"

                    className="inline-flex items-center gap-2 px-4 py-2 bg-blue-600 text-white rounded-lg hover:bg-blue-700 cursor-pointer transition-colors"

                  >

                    <Upload className="w-4 h-4" />

                    Choose CSV File

                  </label>

                </div>

              </div>

              <div className="bg-gray-50 border border-gray-200 rounded-lg p-4">

                <h4 className="font-medium text-gray-900 mb-2">CSV Format Required:</h4>

                <p className="text-sm text-gray-800 mb-2">Name, RollNumber, Year, Section</p>

                <button

                  onClick={downloadSampleCSV}

                  className="inline-flex items-center gap-2 text-sm text-blue-600 hover:text-blue-800"

                >

                  <Download className="w-4 h-4" />

                  Download Sample CSV

                </button>

              </div>

              {csvStudents.length > 0 && (

                <div className="bg-green-50 border border-green-200 rounded-lg p-4">

                  <p className="text-green-800 font-medium">

                    ✓ Successfully loaded {csvStudents.length} students

                  </p>

                  <div className="mt-2 max-h-32 overflow-y-auto">

                    <div className="text-sm text-green-700">

                      {csvStudents.slice(0, 5).map((student, index) => (

                        <div key={index}>

                          {student.name} ({student.rollNumber}) - Year {student.year}, Section {student.section}

                        </div>

                      ))}

                      {csvStudents.length > 5 && <div>... and {csvStudents.length - 5} more</div>}

                    </div>

                  </div>

                  <button

                    onClick={() => setStep('config')}

                    disabled={!hasValidExamSelection}

                    className="mt-3 px-4 py-2 bg-green-600 text-white rounded-lg hover:bg-green-700 disabled:opacity-50 disabled:cursor-not-allowed transition-colors"

                  >

                    Continue to Configuration

                  </button>

                </div>

              )}

            </div>

          </div>

        </div>

      )}

      {/\* Step 2: Configure Classrooms \*/}

      {step === 'config' && (

        <div className="max-w-4xl mx-auto">

          <div className="bg-white rounded-xl shadow-sm border border-gray-200 p-6">

            <div className="flex items-center justify-between mb-4">

              <h2 className="text-lg font-semibold text-gray-900">Configure Classrooms</h2>

              <button

                onClick={addClassroom}

                disabled={selectedClassrooms.length >= classrooms.filter(c => c.available).length}

                className="inline-flex items-center gap-2 px-4 py-2 bg-blue-600 text-white rounded-lg hover:bg-blue-700 disabled:opacity-50 disabled:cursor-not-allowed transition-colors"

              >

                <Plus className="w-4 h-4" />

                Add Classroom

              </button>

            </div>

            {/\* Student Distribution Summary \*/}

            <div className="mb-6 p-4 bg-blue-50 border border-blue-200 rounded-lg">

              <h3 className="font-medium text-blue-900 mb-2">Student Distribution</h3>

              <div className="grid grid-cols-3 gap-4 text-sm">

                <div>

                  <span className="text-blue-700">Total Students:</span>

                  <span className="font-medium text-blue-900 ml-2">{csvStudents.length}</span>

                </div>

                <div>

                  <span className="text-blue-700">Configured Seats:</span>

                  <span className="font-medium text-blue-900 ml-2">{totalConfiguredStudents}</span>

                </div>

                <div>

                  <span className="text-blue-700">Remaining:</span>

                  <span className={`font-medium ml-2 ${csvStudents.length - totalConfiguredStudents < 0 ? 'text-red-600' : 'text-green-600'}`}>

                    {csvStudents.length - totalConfiguredStudents}

                  </span>

                </div>

              </div>

            </div>

            <div className="space-y-6">

              {selectedClassrooms.map((classroom, index) => (

                <div key={index} className="border border-gray-200 rounded-lg p-4">

                  <div className="flex items-center justify-between mb-4">

                    <h3 className="font-medium text-gray-900">

                      Classroom {index + 1} - {classroom.type === 'room' ? 'Room' : 'Seminar Hall'}

                    </h3>

                    <button

                      onClick={() => removeClassroom(index)}

                      className="text-red-600 hover:text-red-800"

                    >

                      <X className="w-4 h-4" />

                    </button>

                  </div>

                  <div className="space-y-4">

                    <div>

                      <label className="block text-sm font-medium text-gray-700 mb-2">

                        Select Classroom

                      </label>

                      <select

                        value={classroom.id}

                        onChange={(e) => changeClassroom(index, e.target.value)}

                        className="w-full px-3 py-2 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500"

                      >

                        {classrooms.filter(c =>

                          c.available && (c.id === classroom.id || !selectedClassrooms.some(sc => sc.id === c.id))

                        ).map(c => (

                          <option key={c.id} value={c.id}>

                            {c.name} ({c.type}) - {c.capacity} capacity

                          </option>

                        ))}

                      </select>

                    </div>

                    {classroom.type === 'room' ? (

                      <div className="grid grid-cols-1 md:grid-cols-4 gap-4">

                        <div>

                          <label className="block text-sm font-medium text-gray-700 mb-1">

                            Students

                          </label>

                          <input

                            type="number"

                            value={classroom.config.numberOfStudents}

                            onChange={(e) => updateClassroomConfig(index, { numberOfStudents: parseInt(e.target.value) || 0 })}

                            className="w-full px-3 py-2 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500"

                            min="0"

                            max={csvStudents.length}

                          />

                        </div>

                        <div>

                          <label className="block text-sm font-medium text-gray-700 mb-1">

                            Benches

                          </label>

                          <input

                            type="number"

                            value={classroom.config.numberOfBenches}

                            onChange={(e) => updateClassroomConfig(index, { numberOfBenches: parseInt(e.target.value) || 0 })}

                            className="w-full px-3 py-2 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500"

                            min="1"

                          />

                        </div>

                        <div>

                          <label className="block text-sm font-medium text-gray-700 mb-1">

                            Students/Bench

                          </label>

                          <select

                            value={classroom.config.studentsPerBench}

                            onChange={(e) => updateClassroomConfig(index, { studentsPerBench: parseInt(e.target.value) })}

                            className="w-full px-3 py-2 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500"

                          >

                            <option value={1}>1</option>

                            <option value={2}>2</option>

                            <option value={3}>3</option>

                            <option value={4}>4</option>

                          </select>

                        </div>

                        <div className="flex items-end">

                          <div className="text-sm text-gray-600">

                            Capacity: {(classroom.config.numberOfBenches || 0) \* (classroom.config.studentsPerBench || 0)}

                          </div>

                        </div>

                      </div>

                    ) : (

                      <div className="grid grid-cols-1 md:grid-cols-4 gap-4">

                        <div>

                          <label className="block text-sm font-medium text-gray-700 mb-1">

                            Students

                          </label>

                          <input

                            type="number"

                            value={classroom.config.numberOfStudents}

                            onChange={(e) => updateClassroomConfig(index, { numberOfStudents: parseInt(e.target.value) || 0 })}

                            className="w-full px-3 py-2 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500"

                            min="0"

                            max={csvStudents.length}

                          />

                        </div>

                        <div>

                          <label className="block text-sm font-medium text-gray-700 mb-1">

                            Rows

                          </label>

                          <input

                            type="number"

                            value={classroom.config.numberOfRows}

                            onChange={(e) => updateClassroomConfig(index, { numberOfRows: parseInt(e.target.value) || 0 })}

                            className="w-full px-3 py-2 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500"

                            min="1"

                          />

                        </div>

                        <div>

                          <label className="block text-sm font-medium text-gray-700 mb-1">

                            Columns

                          </label>

                          <input

                            type="number"

                            value={classroom.config.numberOfColumns}

                            onChange={(e) => updateClassroomConfig(index, { numberOfColumns: parseInt(e.target.value) || 0 })}

                            className="w-full px-3 py-2 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500"

                            min="1"

                          />

                        </div>

                        <div className="flex items-end">

                          <div className="text-sm text-gray-600">

                            Capacity: {(classroom.config.numberOfRows || 0) \* (classroom.config.numberOfColumns || 0)}

                          </div>

                        </div>

                      </div>

                    )}

                  </div>

                </div>

              ))}

              {selectedClassrooms.length === 0 && (

                <div className="text-center py-8 text-gray-500">

                  <Building2 className="w-12 h-12 text-gray-300 mx-auto mb-4" />

                  <p>No classrooms selected. Click "Add Classroom" to start.</p>

                </div>

              )}

            </div>

            <div className="flex gap-3 mt-6">

              <button

                onClick={() => setStep('upload')}

                className="px-4 py-2 border border-gray-300 text-gray-700 rounded-lg hover:bg-gray-50 transition-colors"

              >

                Back

              </button>

              <button

                onClick={generateSeatingArrangement}

                disabled={selectedClassrooms.length === 0 || totalConfiguredStudents === 0}

                className="px-4 py-2 bg-blue-600 text-white rounded-lg hover:bg-blue-700 disabled:opacity-50 disabled:cursor-not-allowed transition-colors"

              >

                Generate Seating Arrangement

              </button>

            </div>

          </div>

        </div>

      )}

      {/\* Step 3: Generated Seating \*/}

      {step === 'generate' && generatedSeating.length > 0 && (

        <div className="space-y-6">

          <div className="bg-white rounded-xl shadow-sm border border-gray-200 p-6">

            {/\* Algorithm Visualization \*/}

            {algoInfo && (

              <div className="mb-6">

                <h3 className="font-semibold text-blue-900 mb-2">Algorithm Used: {algoInfo.name}</h3>

                <p className="text-sm text-blue-800 mb-2">{algoInfo.description}</p>

                <div className="flex flex-col md:flex-row gap-6 items-center">

                  <div className="w-full md:w-1/2">

                    <ResponsiveContainer width="100%" height={220}>

                      <BarChart data={[{ name: algoInfo.name, Runtime: algoInfo.runtime }]}

                        margin={{ top: 20, right: 30, left: 0, bottom: 5 }}>

                        <XAxis dataKey="name" />

                        <YAxis label={{ value: 'ms', angle: -90, position: 'insideLeft' }} />

                        <Tooltip />

                        <Legend />

                        <Bar dataKey="Runtime" fill="#8884d8" />

                      </BarChart>

                    </ResponsiveContainer>

                  </div>

                  <div className="w-full md:w-1/2">

                    <div className="bg-blue-50 border border-blue-200 rounded-lg p-4">

                      <div className="font-medium text-blue-900">Time Complexity:</div>

                      <div className="text-blue-800">{algoInfo.complexity}</div>

                      <div className="mt-2 text-xs text-blue-700">(For seminar halls: Graph Coloring, for classrooms: Bench Assignment)</div>

                      <div className="mt-2 text-xs text-blue-900 font-bold">Runtime: {algoInfo.runtime} ms</div>

                    </div>

                  </div>

                </div>

              </div>

            )}

            {/\* Loading Spinner \*/}

            {loading && (

              <div className="flex flex-col items-center justify-center py-12">

                <svg className="animate-spin h-10 w-10 text-blue-600 mb-4" xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24">

                  <circle className="opacity-25" cx="12" cy="12" r="10" stroke="currentColor" strokeWidth="4"></circle>

                  <path className="opacity-75" fill="currentColor" d="M4 12a8 8 0 018-8v8z"></path>

                </svg>

                <div className="text-blue-700 font-medium">Generating seating arrangement using {algoInfo ? algoInfo.name : 'selected algorithm'}...</div>

              </div>

            )}

            <div className="flex justify-between items-center mb-6">

              <div>

                <h2 className="text-lg font-semibold text-gray-900">Generated Seating Arrangement</h2>

                <p className="text-gray-600">

                  Multi-Classroom Arrangement - {selectedClassrooms.length} classroom(s)

                </p>

              </div>

              <div className="flex gap-2">

                <button

                  onClick={() => downloadSeatingArrangement('csv')}

                  className="inline-flex items-center gap-2 px-4 py-2 bg-green-600 text-white rounded-lg hover:bg-green-700 transition-colors"

                >

                  <Download className="w-4 h-4" />

                  Download CSV

                </button>

                <button

                  onClick={() => downloadSeatingArrangement('pdf')}

                  className="inline-flex items-center gap-2 px-4 py-2 bg-red-600 text-white rounded-lg hover:bg-red-700 transition-colors"

                >

                  <Download className="w-4 h-4" />

                  Download PDF

                </button>

                <button

                  onClick={handleSendMail}

                  className="inline-flex items-center gap-2 px-4 py-2 bg-blue-600 text-white rounded-lg hover:bg-blue-700 transition-colors"

                >

                  <Mail className="w-4 h-4" />

                  Send Mail

                </button>

                <button

                  onClick={saveSeatingArrangement}

                  className="inline-flex items-center gap-2 px-4 py-2 bg-purple-600 text-white rounded-lg hover:bg-purple-700 transition-colors"

                >

                  Save Arrangement

                </button>

              </div>

            </div>

            {/\* Classroom-wise Seating Visualization \*/}

            <div className="space-y-8">

              {selectedClassrooms.map((classroom) => {

                const classroomSeating = generatedSeating.filter(seat => seat.classroomId === classroom.id);

                return (

                  <div key={classroom.id} className="border border-gray-200 rounded-lg p-6">

                    <h3 className="text-lg font-semibold text-gray-900 mb-4">

                      {classroom.name} ({classroom.type === 'room' ? 'Room' : 'Seminar Hall'})

                    </h3>

                    {classroom.type === 'room' ? (

                      <div className="grid gap-2" style={{

                        gridTemplateColumns: `repeat(${classroom.config.studentsPerBench}, 1fr)`,

                        maxWidth: '800px'

                      }}>

                        {Array.from({ length: classroom.config.numberOfBenches! }, (\_, benchIndex) => {

                          const benchNumber = benchIndex + 1;

                          return Array.from({ length: classroom.config.studentsPerBench! }, (\_, seatIndex) => {

                            const seatNumber = seatIndex + 1;

                            const seat = classroomSeating.find(s =>

                              s.benchNumber === benchNumber && s.seatNumber === seatNumber

                            );

                            return (

                              <div

                                key={`B${benchNumber}S${seatNumber}`}

                                className={`p-3 border rounded-lg text-center text-sm ${

                                  seat?.student

                                    ? 'bg-blue-50 border-blue-200'

                                    : 'bg-gray-50 border-gray-200'

                                }`}

                              >

                                <div className="font-medium text-xs text-gray-600 mb-1">

                                  B{benchNumber}S{seatNumber}

                                </div>

                                {seat?.student ? (

                                  <div>

                                    <div className="font-medium truncate">{seat.student.name}</div>

                                    <div className="text-xs text-gray-600">{seat.student.rollNumber}</div>

                                    <div className="text-xs text-gray-600">Y{seat.student.year} {seat.student.section}</div>

                                  </div>

                                ) : (

                                  <div className="text-gray-400 text-xs">Empty</div>

                                )}

                              </div>

                            );

                          });

                        })}

                      </div>

                    ) : (

                      <div className="grid gap-2" style={{

                        gridTemplateColumns: `repeat(${classroom.config.numberOfColumns}, 1fr)`,

                        maxWidth: '1000px'

                      }}>

                        {Array.from({ length: classroom.config.numberOfRows! }, (\_, rowIndex) => {

                          const row = rowIndex + 1;

                          return Array.from({ length: classroom.config.numberOfColumns! }, (\_, colIndex) => {

                            const col = colIndex + 1;

                            const seat = classroomSeating.find(s =>

                              s.row === row && s.column === col

                            );

                            return (

                              <div

                                key={`R${row}C${col}`}

                                className={`p-2 border rounded text-center text-xs ${

                                  seat?.student

                                    ? 'bg-blue-50 border-blue-200'

                                    : 'bg-gray-50 border-gray-200'

                                }`}

                              >

                                <div className="font-medium text-xs text-gray-600 mb-1">

                                  R{row}C{col}

                                </div>

                                {seat?.student ? (

                                  <div>

                                    <div className="font-medium truncate text-xs">{seat.student.name}</div>

                                    <div className="text-xs text-gray-600">{seat.student.rollNumber}</div>

                                    <div className="text-xs text-gray-600">Y{seat.student.year} {seat.student.section}</div>

                                  </div>

                                ) : (

                                  <div className="text-gray-400 text-xs">Empty</div>

                                )}

                              </div>

                            );

                          });

                        })}

                      </div>

                    )}

                    {/\* Classroom Summary \*/}

                    <div className="mt-4 bg-gray-50 rounded-lg p-3">

                      <div className="grid grid-cols-4 gap-4 text-sm">

                        <div>

                          <div className="text-gray-600">Total Seats</div>

                          <div className="font-medium">

                            {classroom.type === 'room'

                              ? (classroom.config.numberOfBenches! \* classroom.config.studentsPerBench!)

                              : (classroom.config.numberOfRows! \* classroom.config.numberOfColumns!)

                            }

                          </div>

                        </div>

                        <div>

                          <div className="text-gray-600">Occupied</div>

                          <div className="font-medium">{classroomSeating.filter(s => s.student).length}</div>

                        </div>

                        <div>

                          <div className="text-gray-600">Empty</div>

                          <div className="font-medium">{classroomSeating.filter(s => !s.student).length}</div>

                        </div>

                        <div>

                          <div className="text-gray-600">Utilization</div>

                          <div className="font-medium">

                            {Math.round((classroomSeating.filter(s => s.student).length / classroomSeating.length) \* 100)}%

                          </div>

                        </div>

                      </div>

                    </div>

                  </div>

                );

              })}

            </div>

            {/\* Overall Summary \*/}

            <div className="bg-gray-50 rounded-lg p-4 mt-6">

              <h4 className="font-medium text-gray-900 mb-3">Overall Summary</h4>

              <div className="grid grid-cols-2 md:grid-cols-4 gap-4 text-sm">

                <div>

                  <div className="text-gray-600">Total Students</div>

                  <div className="font-medium">{csvStudents.length}</div>

                </div>

                <div>

                  <div className="text-gray-600">Students Seated</div>

                  <div className="font-medium">{generatedSeating.filter(s => s.student).length}</div>

                </div>

                <div>

                  <div className="text-gray-600">Classrooms Used</div>

                  <div className="font-medium">{selectedClassrooms.length}</div>

                </div>

                <div>

                  <div className="text-gray-600">Overall Utilization</div>

                  <div className="font-medium">

                    {Math.round((generatedSeating.filter(s => s.student).length / generatedSeating.length) \* 100)}%

                  </div>

                </div>

              </div>

            </div>

            <div className="flex gap-3 mt-6">

              <button

                onClick={() => setStep('config')}

                className="px-4 py-2 border border-gray-300 text-gray-700 rounded-lg hover:bg-gray-50 transition-colors"

              >

                Back to Configuration

              </button>

              <button

                onClick={() => {

                  setStep('upload');

                  setCsvStudents([]);

                  setGeneratedSeating([]);

                  setExamSelections({ year1Exam: '', year2Exam: '', year3Exam: '', year4Exam: '' });

                  setSelectedClassrooms([]);

                }}

                className="px-4 py-2 border border-gray-300 text-gray-700 rounded-lg hover:bg-gray-50 transition-colors"

              >

                Start New Arrangement

              </button>

            </div>

          </div>

        </div>

      )}

    </div>

  );

};

export default SeatingArrangements;